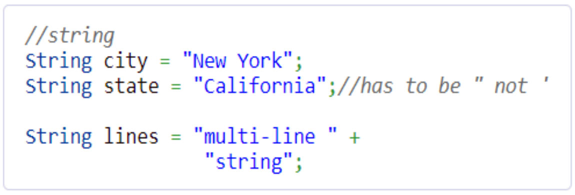
JAVA vs PYTHON (Datatypes)

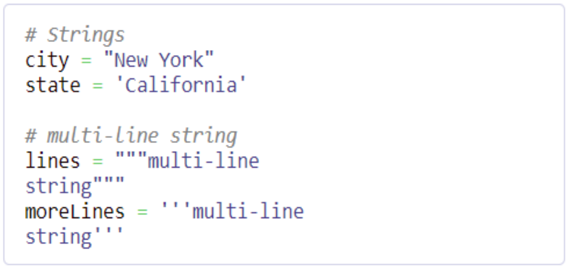
1.) Strings

Java



```java
//string
String city = "New York";
String state = "California";//has to be " not '

String lines = "multi-line " +
               "string";
```

Python

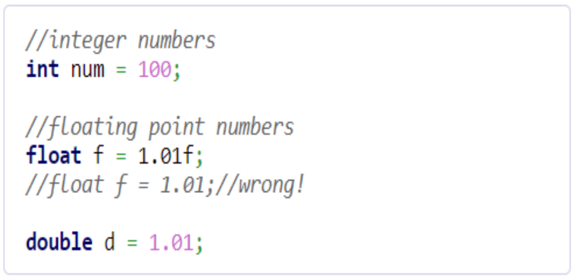


```python
# Strings
city = "New York"
state = 'California'

# multi-line string
lines = """multi-line
string"""
moreLines = '''multi-line
string'''
```

In python, string can reside in a pair of single quotes as well as a pair of double quotes. It supports multiplication: "x"*3 is "xxx".

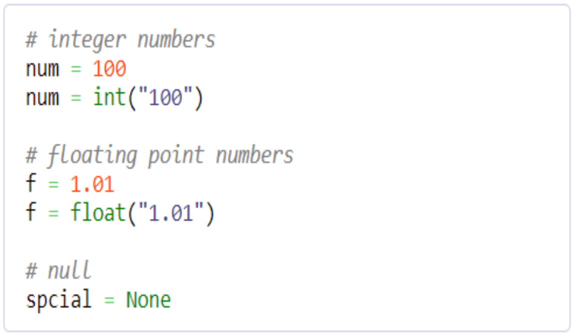
2.) Numbers

Java



```java
//integer numbers
int num = 100;

//floating point numbers
float f = 1.01f;
//float f = 1.01;//wrong!

double d = 1.01;
```

Python



```python
# integer numbers
num = 100
num = int("100")

# floating point numbers
f = 1.01
f = float("1.01")

# null
spcial = None
```

In Java, when you type something like 1.01, its interpreted as a double. Double is a 64-bit precision IEEE 754 floating point, while float is a 32-bit precision IEEE 754 floating point. As a float is less precise than a double, the conversion cannot be performed implicitly.

3.) Null

Java

```
//null
Object special = null;
```

Python

```
# null
spcial = None
```

4.) Lists

Java

```
//arraylist is closest with list in python
ArrayList<Integer> aList =
  new ArrayList<Integer>();

//add
aList.add(1);
aList.add(3);
aList.add(2);
aList.add(4);

//index
int n = aList.get(0);

//get sub list
List<Integer> subList =
  aList.subList(0, 2);
//1, 3
```

Python

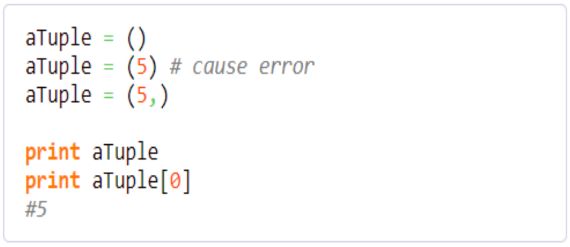
```
aList = []
aList = [1, 'mike', 'john']

#append
aList.append(2)

# extend
aList.extend(["new","list"])

print aList
#[1, 'mike', 'john', 2, 'new', 'list']

aList = [0,1,2,3,4,5,6]
# size
print len(aList)
#7

print aList[2]
#2

print aList[0:3]
#[0, 1, 2]

print aList[2:]
#[2, 3, 4, 5, 6]

print aList[-2]
#5

#lists are mutable
aList[0] = 10
print aList
#[10, 1, 2, 3, 4, 5, 6]
```

5.) Tuples

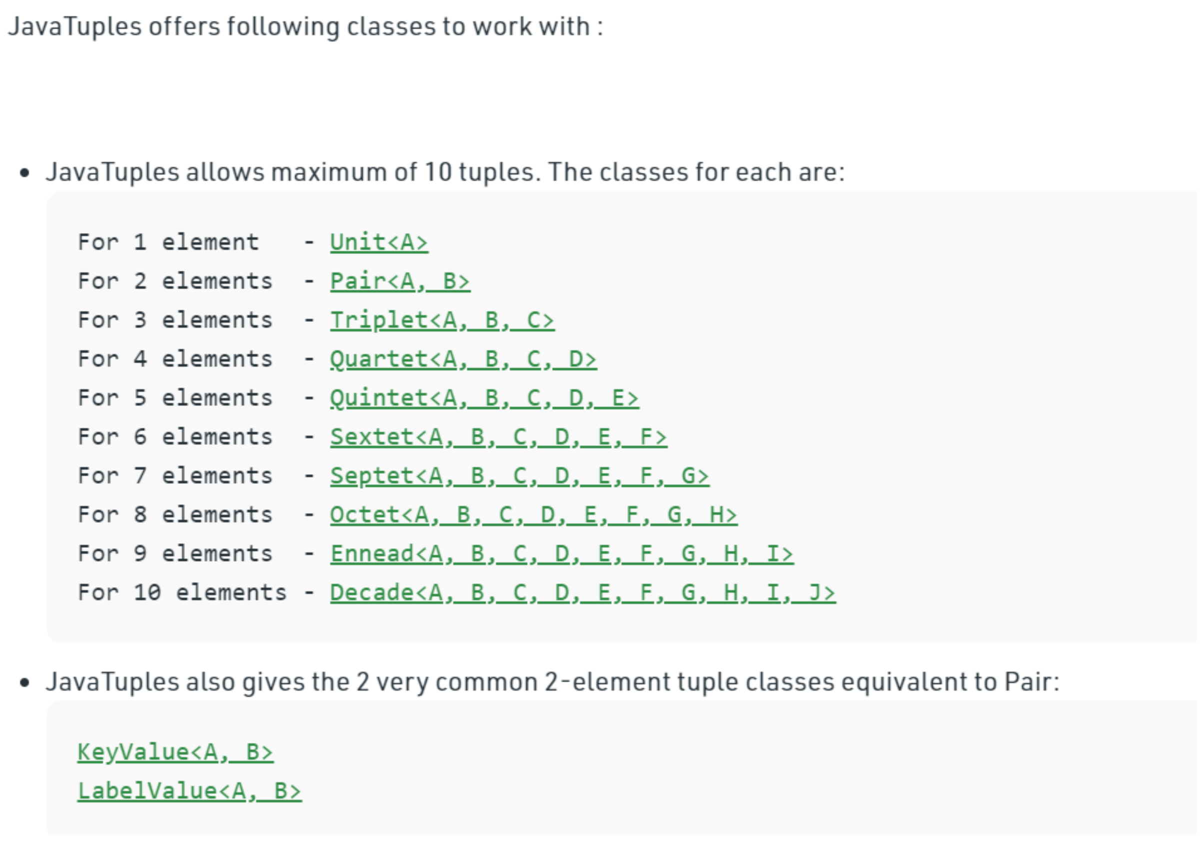
Java

JavaTuples is a Java library that offers classes, functions and data structures to work with tuples. It is one of the simplest java library ever made.

Python



```python
aTuple = ()
aTuple = (5) # cause error
aTuple = (5,)

print aTuple
print aTuple[0]
#5
```



JavaTuples offers following classes to work with :

- JavaTuples allows maximum of 10 tuples. The classes for each are:

```
For 1 element   - Unit<A>
For 2 elements  - Pair<A, B>
For 3 elements  - Triplet<A, B, C>
For 4 elements  - Quartet<A, B, C, D>
For 5 elements  - Quintet<A, B, C, D, E>
For 6 elements  - Sextet<A, B, C, D, E, F>
For 7 elements  - Septet<A, B, C, D, E, F, G>
For 8 elements  - Octet<A, B, C, D, E, F, G, H>
For 9 elements  - Ennead<A, B, C, D, E, F, G, H, I>
For 10 elements - Decade<A, B, C, D, E, F, G, H, I, J>
```

- JavaTuples also gives the 2 very common 2-element tuple classes equivalent to Pair:

```
KeyValue<A, B>
LabelValue<A, B>
```

Advantage of using JavaTuples over Lists/Arrays:

Think of a scenario where you want to store the details of a student in just one entity, like Name, Roll Number, Father’s Name, Contact Number. Now the most common approach that strikes the mind is to construct a data structure that would take the fields as required. This is where Tuples come into play. With Tuples, any separate data structure need not to be created. Instead, for this scenario, a Quartet<A, B, C, D> can be simply used.

Therefore, common data structures like List, Array :

- Can be of a specific type only.
- Can be of infinte elements.

Whereas, Tuples :

- Can be of any type, yet they are typesafe
- Can be of limited number only, from 1-10

Example to create a pair and sextet JavaTuples:

Example:

```
Pair<Integer, String> pair = new Pair<Integer, String>(
                              Integer.valurOf(1), "Geeks");

Sextet<Integer, String, Integer, String, Integer, String> sextet
                           = new Sextet<Integer, String, Integer,
                                        String, Integer, String>(
                                            Integer.valueOf(1), "Geeks",
                                            Integer.valueOf(2), "For",
                                            Integer.valueOf(3), "Geeks"
                                            );
```

Getting JavaTuples values

- **Syntax**:

```
NthTuple<type 1, type 2, .., type n> nthTuple = new NthTuple
                <type 1, type 2, .., type n>(value 1, value 2, .., value n);
typeX valX = nthTuple.getValueX-1();
```

- **Example**:

```
Pair < Integer, String >
pair = new Pair<Integer, String>(Integer.valurOf(1), "Geeks");

// This will set val2 = "Geeks"
String val2 = pair.getValue1();

Sextet<Integer, String, Integer, String, Integer, String> sextet
    = new Sextet<Integer, String, Integer, String, Integer, String>(
                                        Integer.valueOf(1), "Geeks",
                                        Integer.valueOf(2), "For",
                                        Integer.valueOf(3), "Geeks"
                                        );

// This will set val5 = 3
Integer val5 = sextet.getValue4();
```

In Python, tuples are similar with lists, and the difference between them is that tuple is immutable. That means methods that change lists' value can not be used on tuples.

To assign a single element tuple, it has to be: aTuple = (5,). If comma is removed, it will be wrong.

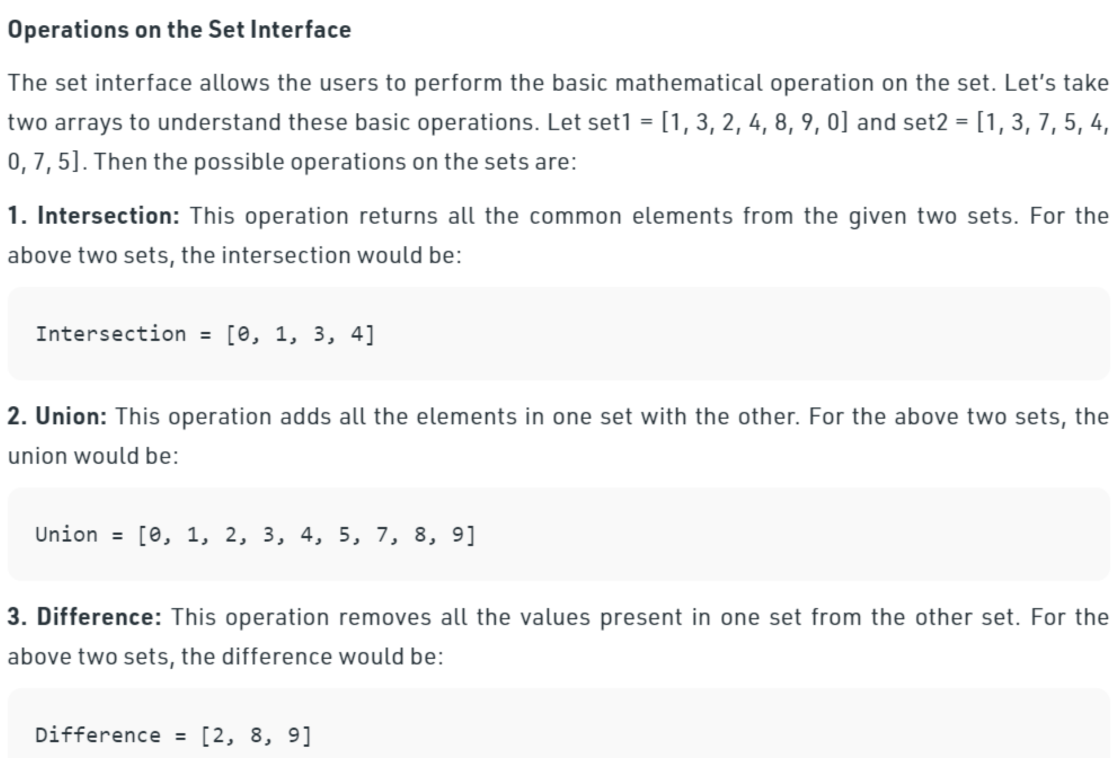
6.) <u>Sets</u>

Java

```java
//hashset
HashSet<String> aSet = new HashSet<String>();
aSet.add("aaaa");
aSet.add("bbbb");
aSet.add("cccc");
aSet.add("dddd");

//iterate over set
Iterator<String> iterator = aSet.iterator();
while (iterator.hasNext()) {
        System.out.print(iterator.next() + " ");
}

HashSet<String> bSet = new HashSet<String>();
bSet.add("eeee");
bSet.add("ffff");
bSet.add("gggg");
bSet.add("dddd");

//check if bSet is a subset of aSet
boolean b = aSet.containsAll(bSet);

//union - transform aSet
//into the union of aSet and bSet
aSet.addAll(bSet);

//intersection - transforms aSet
//into the intersection of aSet and bSet
aSet.retainAll(bSet);

//difference - transforms aSet
//into the (asymmetric) set difference
// of aSet and bSet.
aSet.removeAll(bSet);
```

Python

```python
aSet = set()
aSet = set("one") # a set containing three letters
#set(['e', 'o', 'n'])

aSet = set(['one', 'two', 'three'])
#set(['three', 'two', 'one'])
#a set containing three words

#iterate over set
for v in aSet:
    print v

bSet = set(['three','four', 'five'])

#union
cSet = aSet | bSet
#set(['four', 'one', 'five', 'three', 'two'])

#intersection
dSet = aSet & bSet

#find elements in aSet not bSet
eSet = aSet.difference(bSet)

#add element
bSet.add("six")
#set(['four', 'six', 'five', 'three'])
```



Operations on the Set Interface

The set interface allows the users to perform the basic mathematical operation on the set. Let's take two arrays to understand these basic operations. Let set1 = [1, 3, 2, 4, 8, 9, 0] and set2 = [1, 3, 7, 5, 4, 0, 7, 5]. Then the possible operations on the sets are:

1. Intersection: This operation returns all the common elements from the given two sets. For the above two sets, the intersection would be:

```
Intersection = [0, 1, 3, 4]
```

2. Union: This operation adds all the elements in one set with the other. For the above two sets, the union would be:

```
Union = [0, 1, 2, 3, 4, 5, 7, 8, 9]
```

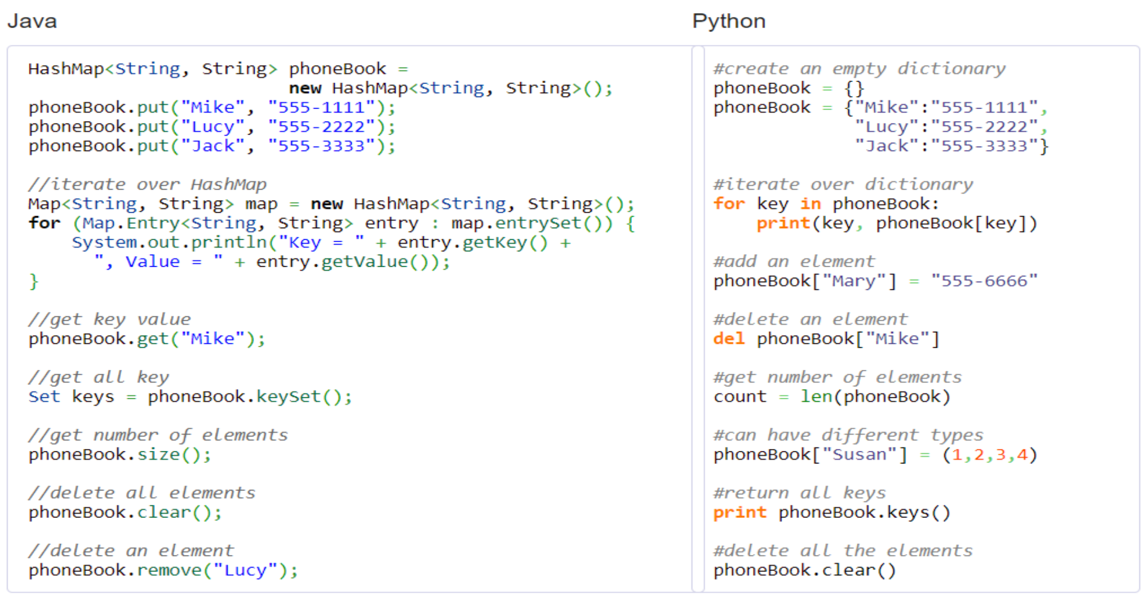
3. Difference: This operation removes all the values present in one set from the other set. For the above two sets, the difference would be:

```
Difference = [2, 8, 9]
```

The set interface present in the java.util package and extends the Collection interface is an unordered collection of objects in which duplicate values cannot be stored. It is an interface which implements the mathematical set.

7.) Dictionaries

Dictionaries in Python is like Maps in Java.



Java

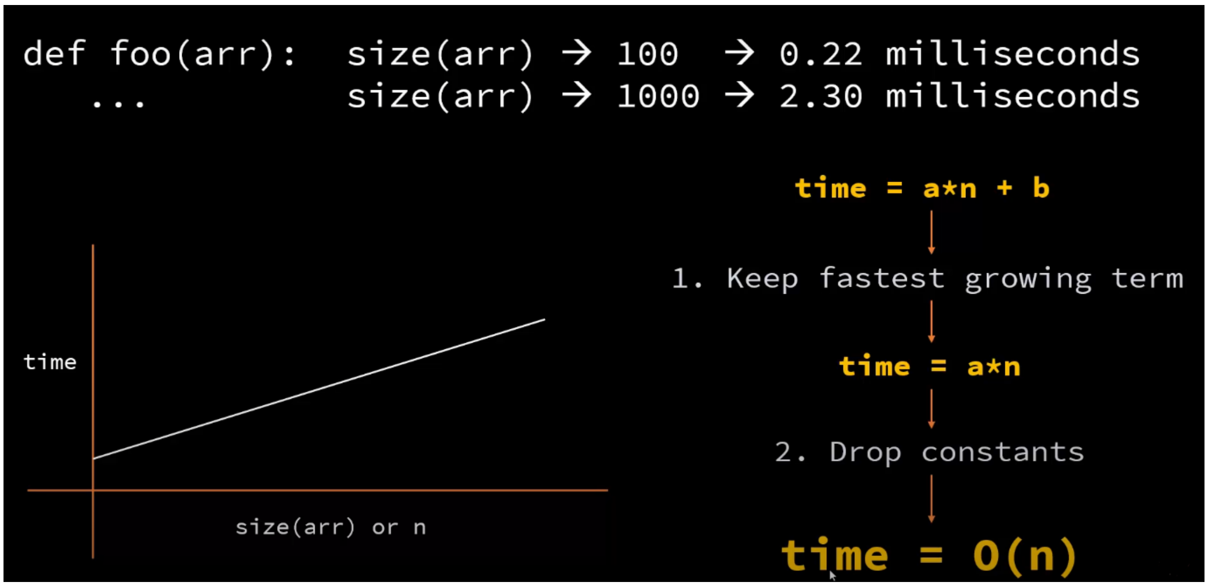
```java
HashMap<String, String> phoneBook =
                        new HashMap<String, String>();
phoneBook.put("Mike", "555-1111");
phoneBook.put("Lucy", "555-2222");
phoneBook.put("Jack", "555-3333");

//iterate over HashMap
Map<String, String> map = new HashMap<String, String>();
for (Map.Entry<String, String> entry : map.entrySet()) {
    System.out.println("Key = " + entry.getKey() +
      ", Value = " + entry.getValue());
}

//get key value
phoneBook.get("Mike");

//get all key
Set keys = phoneBook.keySet();

//get number of elements
phoneBook.size();

//delete all elements
phoneBook.clear();

//delete an element
phoneBook.remove("Lucy");
```

Python

```python
#create an empty dictionary
phoneBook = {}
phoneBook = {"Mike":"555-1111",
             "Lucy":"555-2222",
             "Jack":"555-3333"}

#iterate over dictionary
for key in phoneBook:
    print(key, phoneBook[key])

#add an element
phoneBook["Mary"] = "555-6666"

#delete an element
del phoneBook["Mike"]

#get number of elements
count = len(phoneBook)

#can have different types
phoneBook["Susan"] = (1,2,3,4)

#return all keys
print phoneBook.keys()

#delete all the elements
phoneBook.clear()
```

Big O notation

Big O notation is used to measure how running time or space requirement for your program grows as input size grows.



```
def foo(arr):   size(arr) → 100  → 0.22 milliseconds
   ...          size(arr) → 1000 → 2.30 milliseconds
```

time = a*n + b

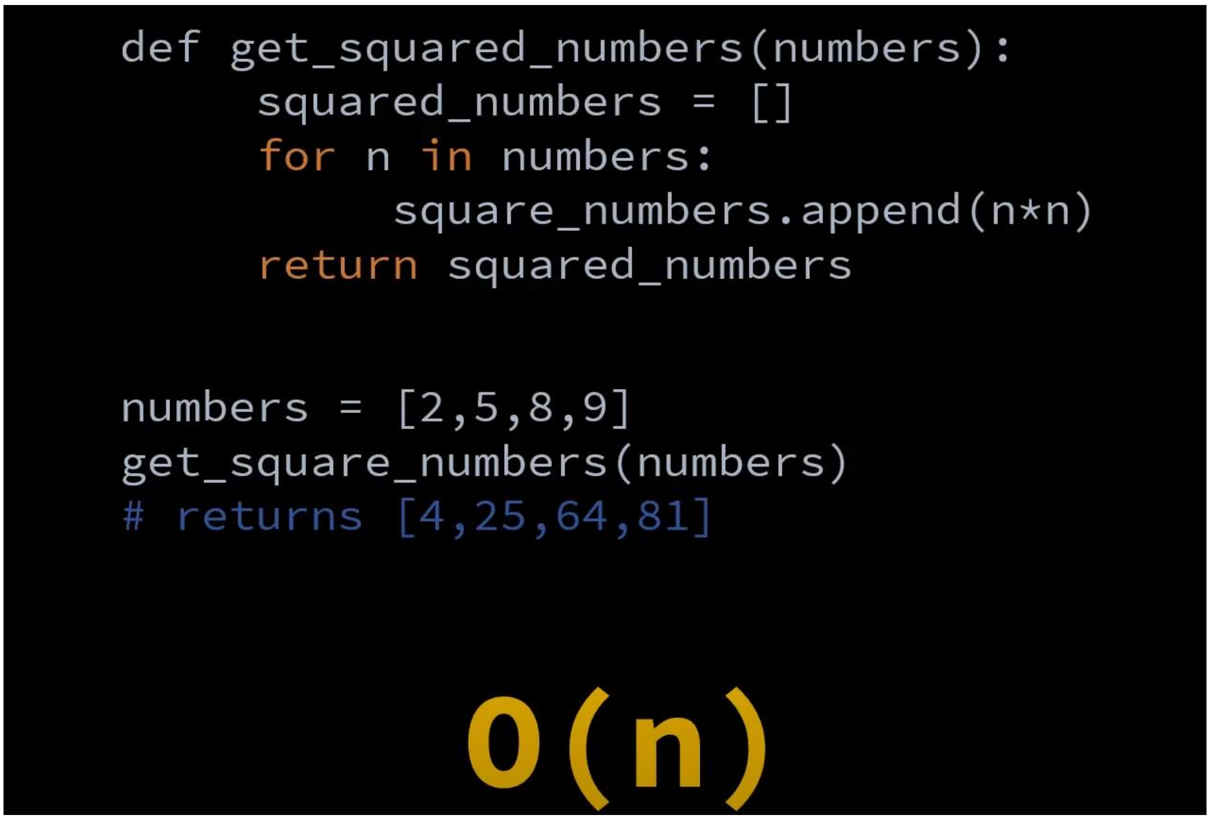
1. Keep fastest growing term

time = a*n

2. Drop constants

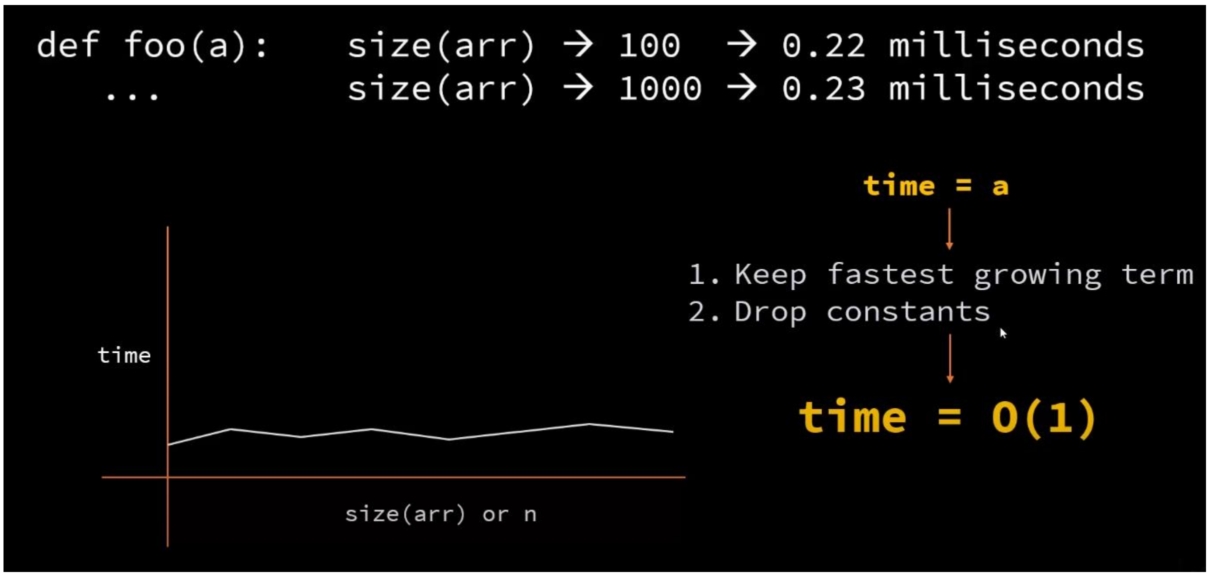
time = O(n)

Example of program of O(n) time complexity:



```
def get_squared_numbers(numbers):
     squared_numbers = []
     for n in numbers:
          square_numbers.append(n*n)
     return squared_numbers


numbers = [2,5,8,9]
get_square_numbers(numbers)
# returns [4,25,64,81]
```

O(n)



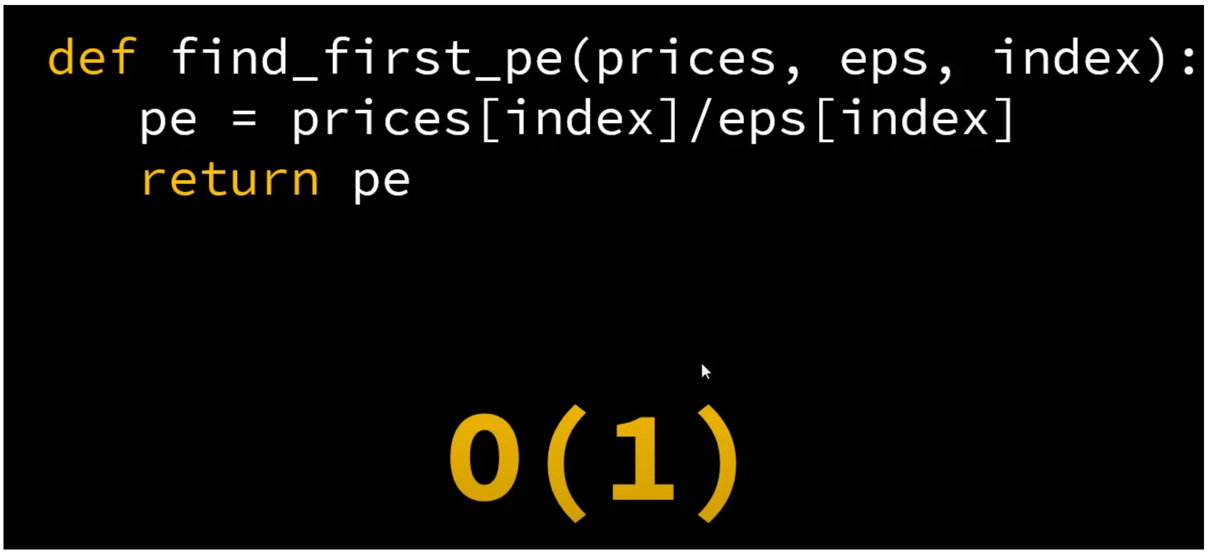
```
def foo(a):      size(arr) → 100  → 0.22 milliseconds
   ...           size(arr) → 1000 → 0.23 milliseconds
```

time = a

1. Keep fastest growing term
2. Drop constants

time = O(1)

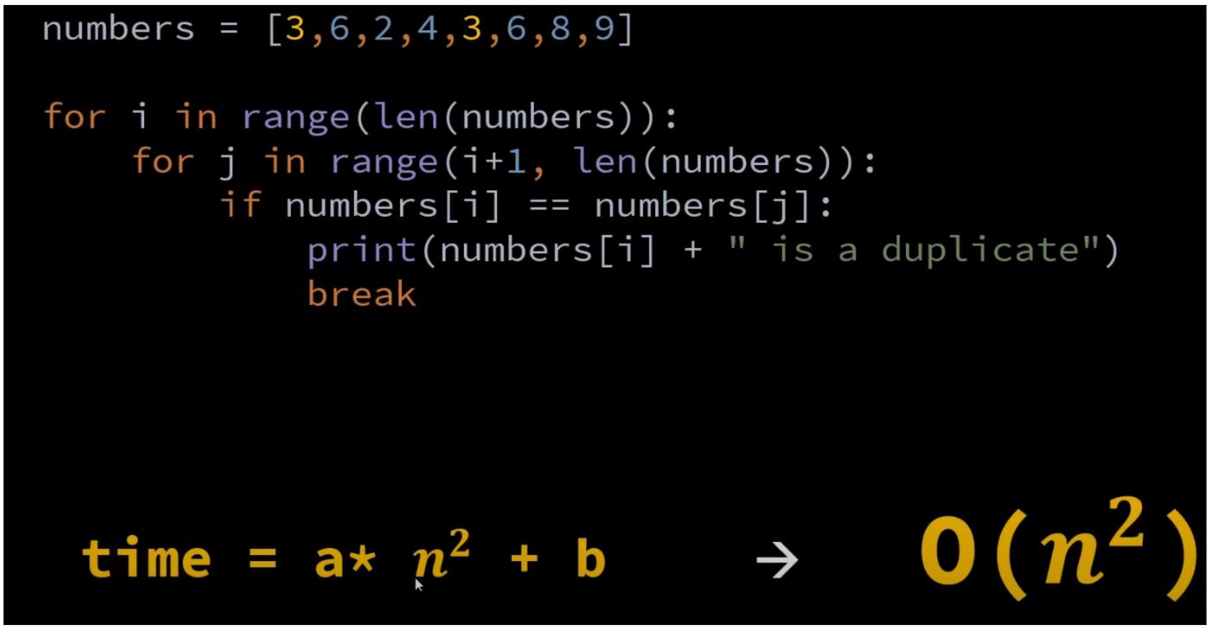
Example of a function whose time complexity is O(1):



```python
def find_first_pe(prices, eps, index):
    pe = prices[index]/eps[index]
    return pe
```

O(1)

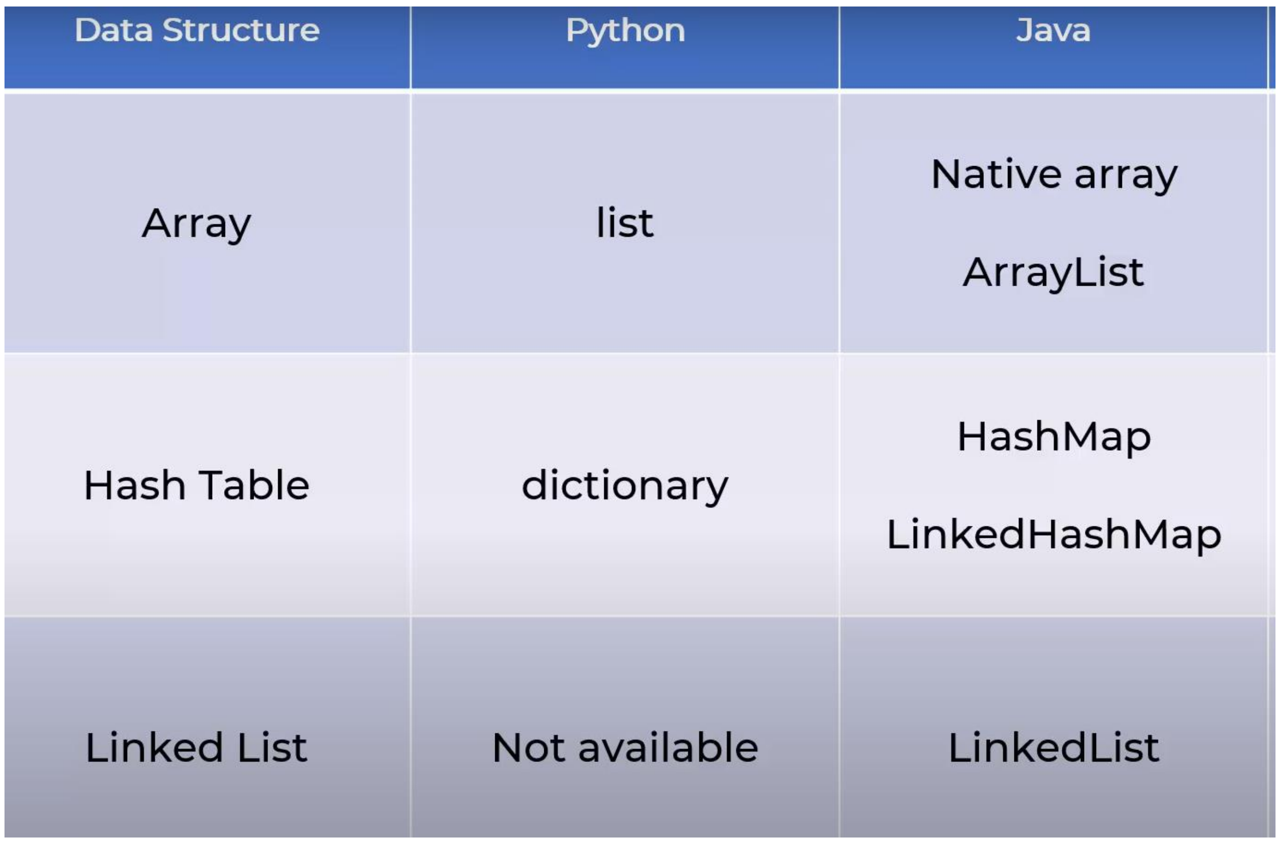
Example of a program whose time complexity is O(n^2):



```
numbers = [3,6,2,4,3,6,8,9]

for i in range(len(numbers)):
    for j in range(i+1, len(numbers)):
        if numbers[i] == numbers[j]:
            print(numbers[i] + " is a duplicate")
            break
```

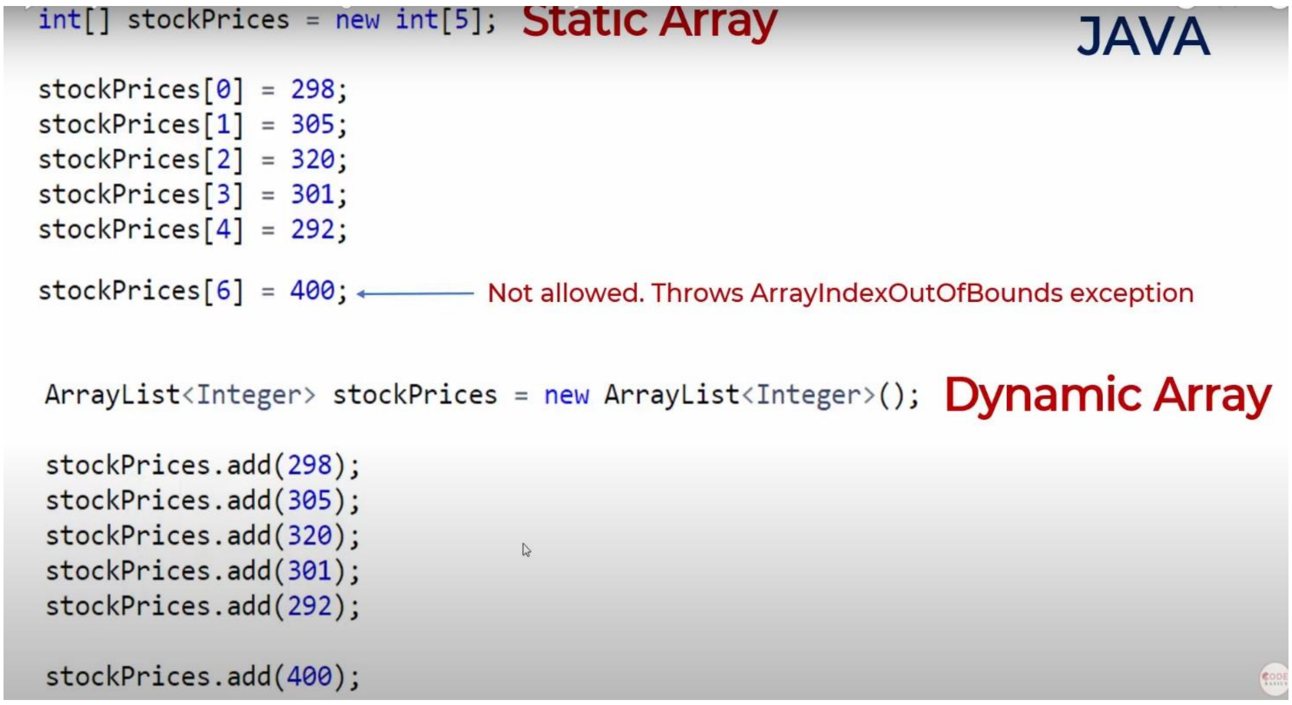
$$time = a* n^2 + b \rightarrow O(n^2)$$

<u>Data Structures in Python Vs Java</u>



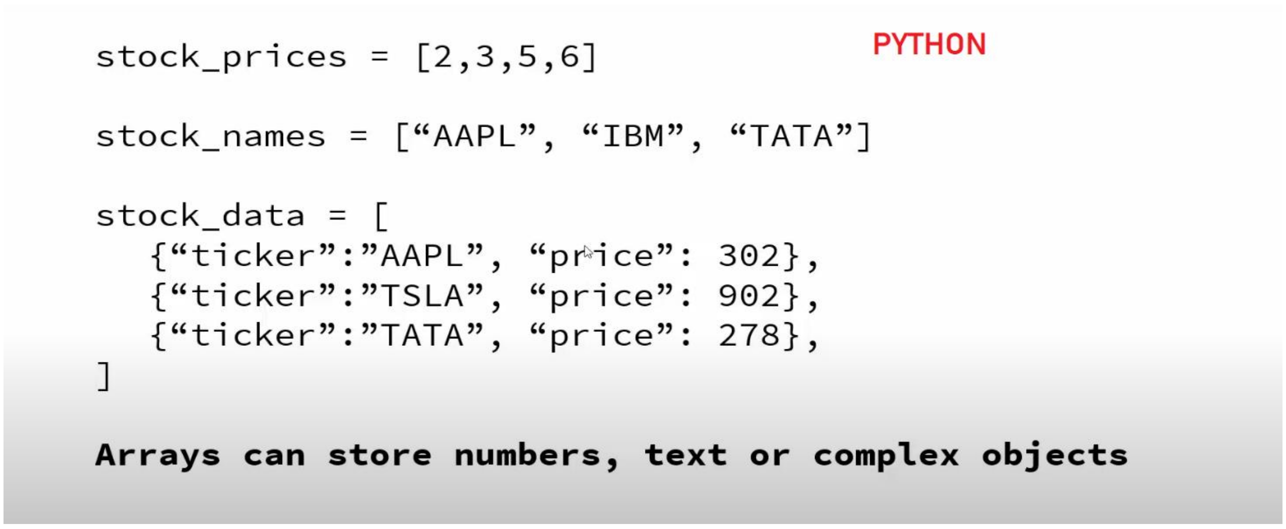
Data Structure	Python	Java
Array	list	Native array ArrayList
Hash Table	dictionary	HashMap LinkedHashMap
Linked List	Not available	LinkedList

1.) Arrays:



```
int[] stockPrices = new int[5];

stockPrices[0] = 298;
stockPrices[1] = 305;
stockPrices[2] = 320;
stockPrices[3] = 301;
stockPrices[4] = 292;

stockPrices[6] = 400;
```

```
ArrayList<Integer> stockPrices = new ArrayList<Integer>();

stockPrices.add(298);
stockPrices.add(305);
stockPrices.add(320);
stockPrices.add(301);
stockPrices.add(292);

stockPrices.add(400);
```

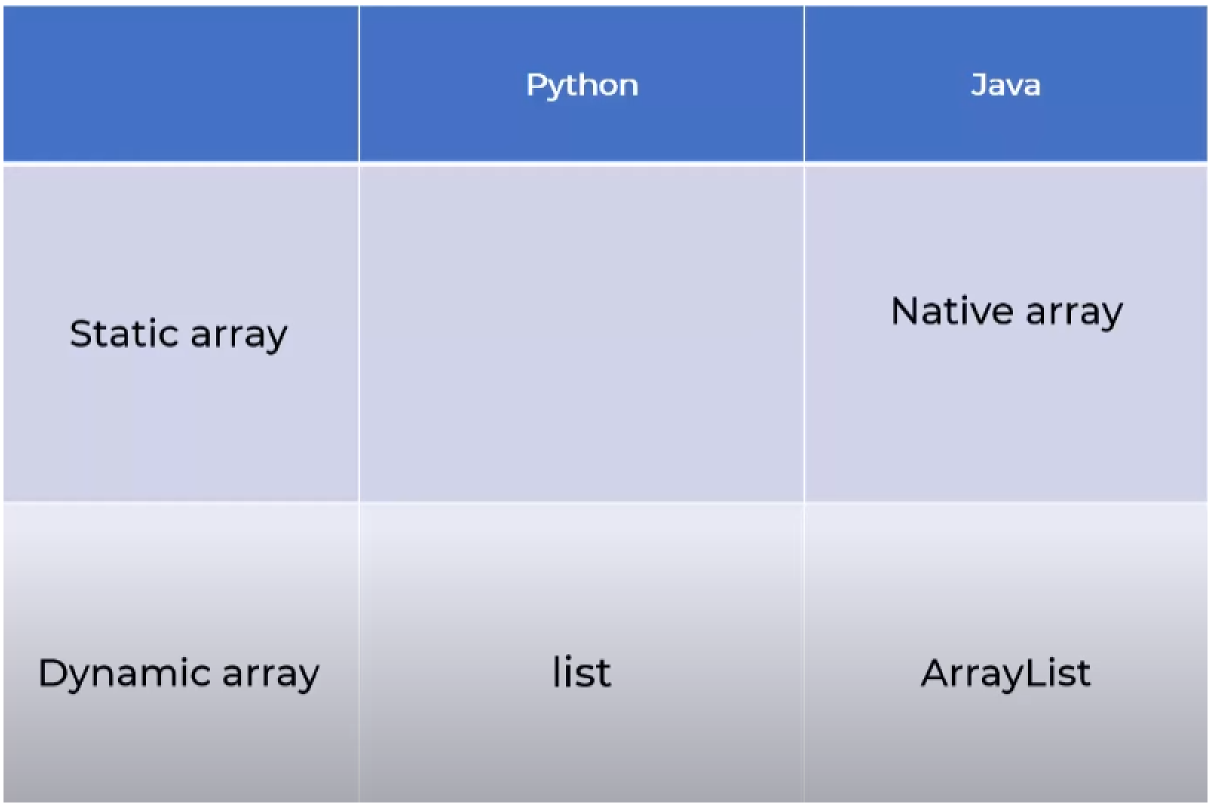


```
stock_prices = [2,3,5,6]

stock_names = [“AAPL”, “IBM”, “TATA”]

stock_data = [
   {“ticker”:”AAPL”, “price”: 302},
   {“ticker”:”TSLA”, “price”: 902},
   {“ticker”:”TATA”, “price”: 278},
]
```

Arrays can store numbers, text or complex objects

Python's lists are implemented with dynamically resized arrays. Python's tuples are implemented with fixed-size arrays. In Java we have static and dynamic arrays both.

There are actually two Python types commonly referred to as arrays, not counting the common newbie usage of "array" to refer to Python lists. There are the arrays provided by the array module, and there are NumPy's ndarrays.



	Python	Java
Static array		Native array
Dynamic array	list	ArrayList

References:

1.) https://www.programcreek.com/2012/09/java-vs-python-data-types/
2.) https://www.youtube.com/watch?v=_t2GVaQasRY&list=PLeo1K3hjS3uu_n_a__MI_KktGTLYopZ12
3.) https://realpython.com/python-data-structures/
4.) https://stackoverflow.com/questions/22124831/data-structures-in-java-python-and-their-implementations-lists-arrays-tuple
5.) https://www.geeksforgeeks.org/javatuples-introduction
6.) https://www.geeksforgeeks.org/set-in-java/